Неочевидное решение очевидной проблемы:

грузим файлы

team

Немного обо мне:

2 года в android разработке

Преподаю в ИТМО

Сегодня в программе:

1

Поговорим о проблемах использования OkHttp

2

Обсудим почему вместо ktor решили писать свой велосипед

3

Вспомним java.io и java.nio

4

Подружим каналы с корутинами и TLS

5

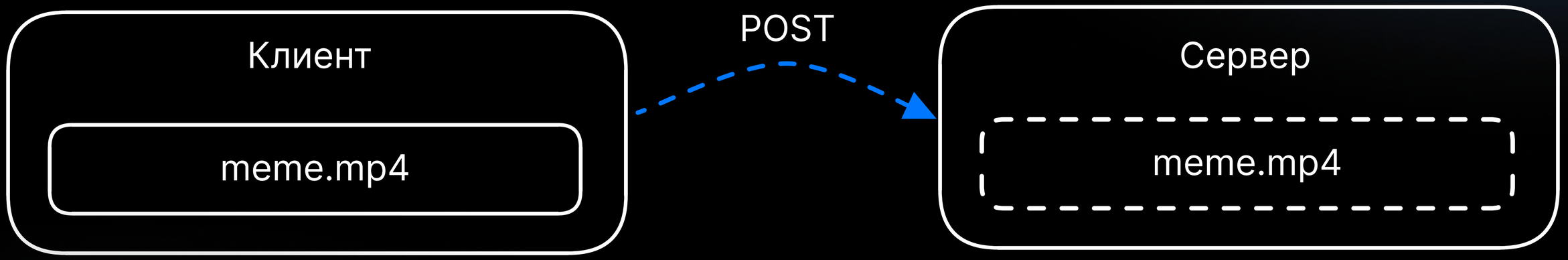
Засетапим универсальный аплоадер

Грузим файлы

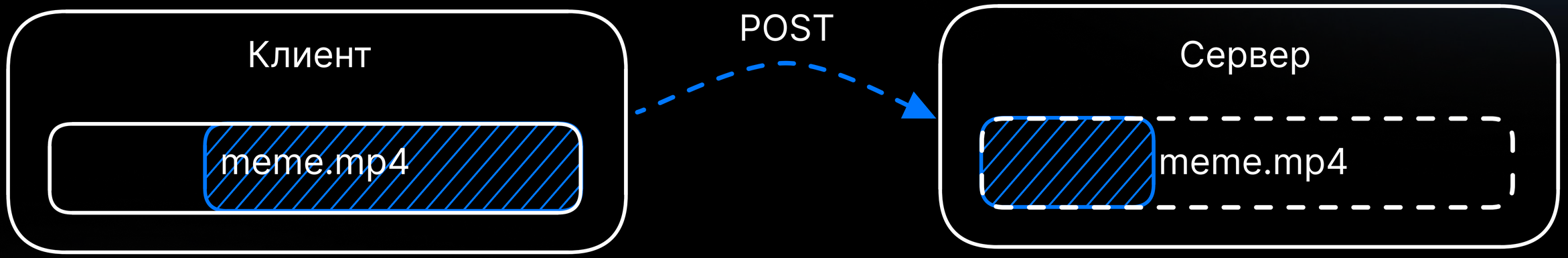
Почему мы вообще занялись этим?

team

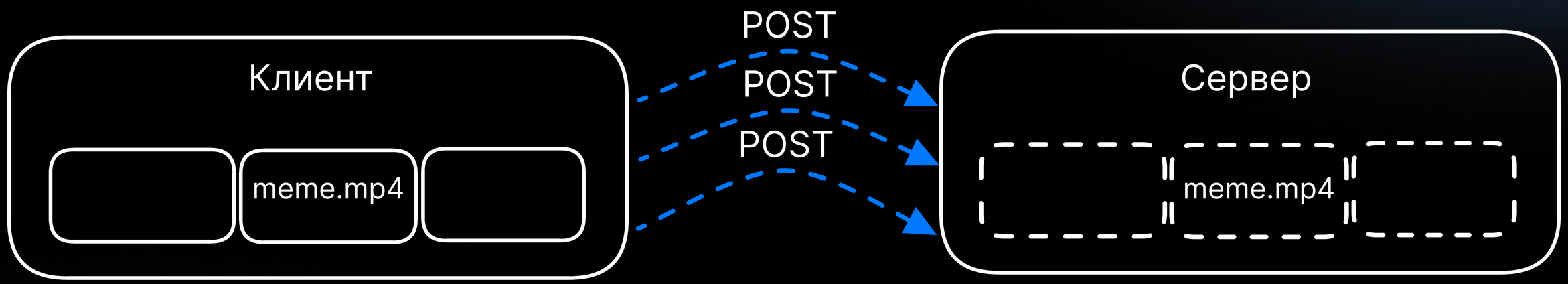
Загрузка как она есть



Обрабатываем обрывы сети



А если чанков будет много?



Требования

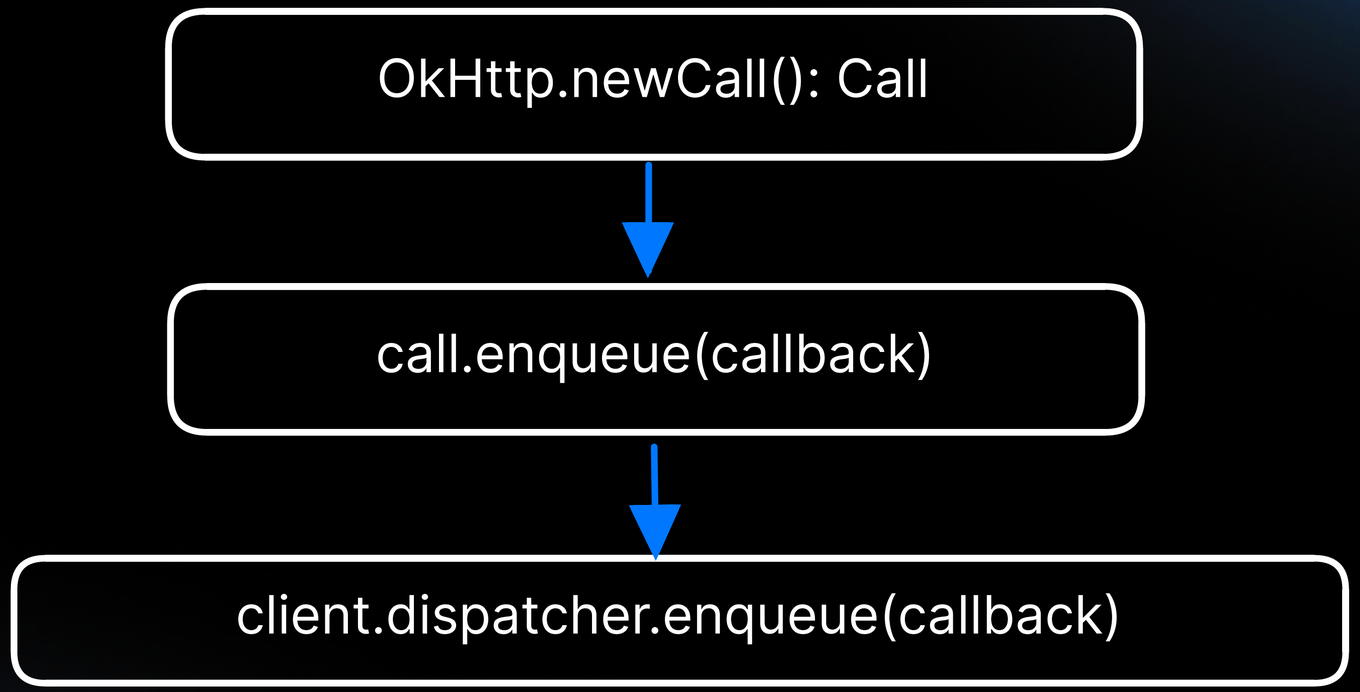
1. Аплоад больших файлов
   до 4х гб
2. Работа на слабых сетях
3. Параллельный аплоад нескольких файлов
4. Параллельный аплоад одного файла по чанкам

Грузим файлы

Что используется сейчас?

team

Как выполняется асинхронный запрос?



okhttp3.Dispatcher.kt

```
@get:Synchronized
val executorService: ExecutorService
 get() {
    if (executorServiceOrNull == null) {
      executorServiceOrNull = ThreadPoolExecutor(0, Int.MAX_VALUE, 60, ...)
    }
    return executorServiceOrNull!!
  }
```

Что если мы явно не передаем executor?

```
@get:Synchronized
val executorService: ExecutorService
 get() {
    if (executorServiceOrNull == null) {
      executorServiceOrNull = ThreadPoolExecutor(0, Int.MAX_VALUE, 60,  ...)
    }
    return executorServiceOrNull!!
  }
```

Грузим файлы

А что плохого в unbounded пуле?

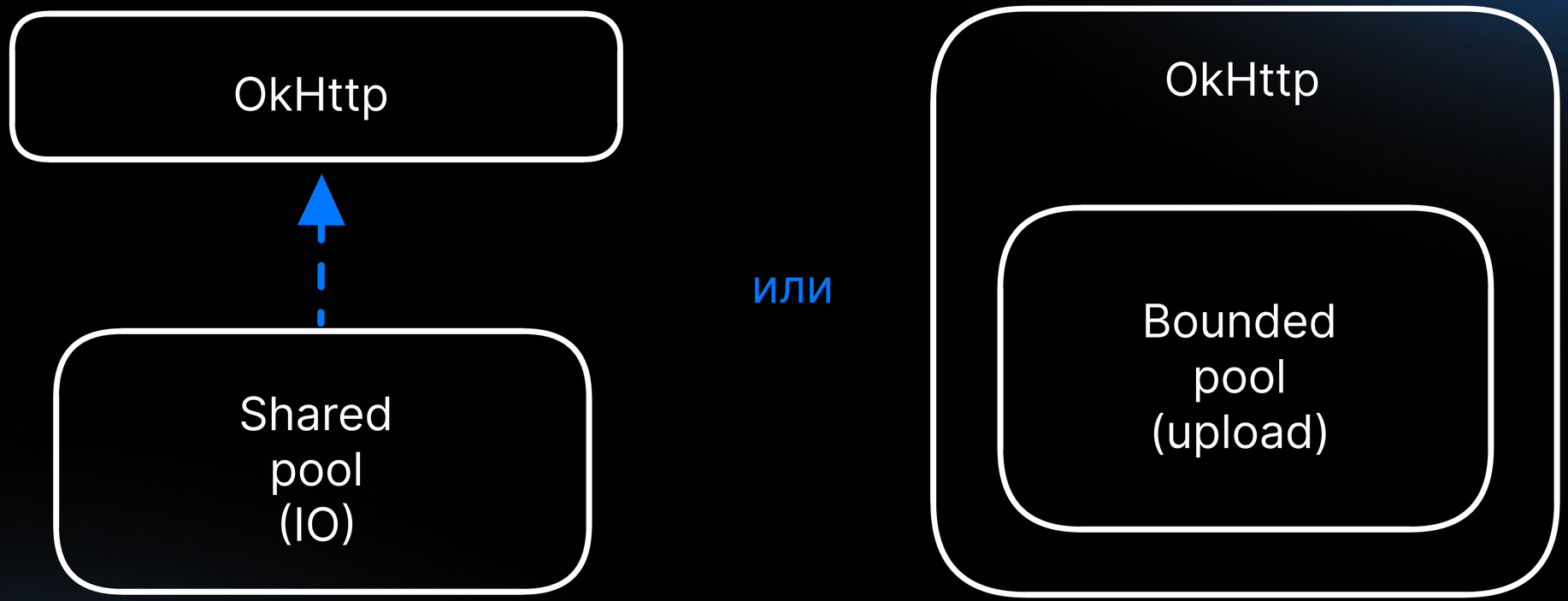
team

Наша борьба

1. Стремимся держать минимально необходимое количество потоков и по максимуму их переиспользовать

2. Аплоадер должен заработать в Single-Core mode

Провайдим свой executor в OkHttp



Грузим файлы

Как реализовать требование к параллельности без unbounded пула?

team

Грузим файлы

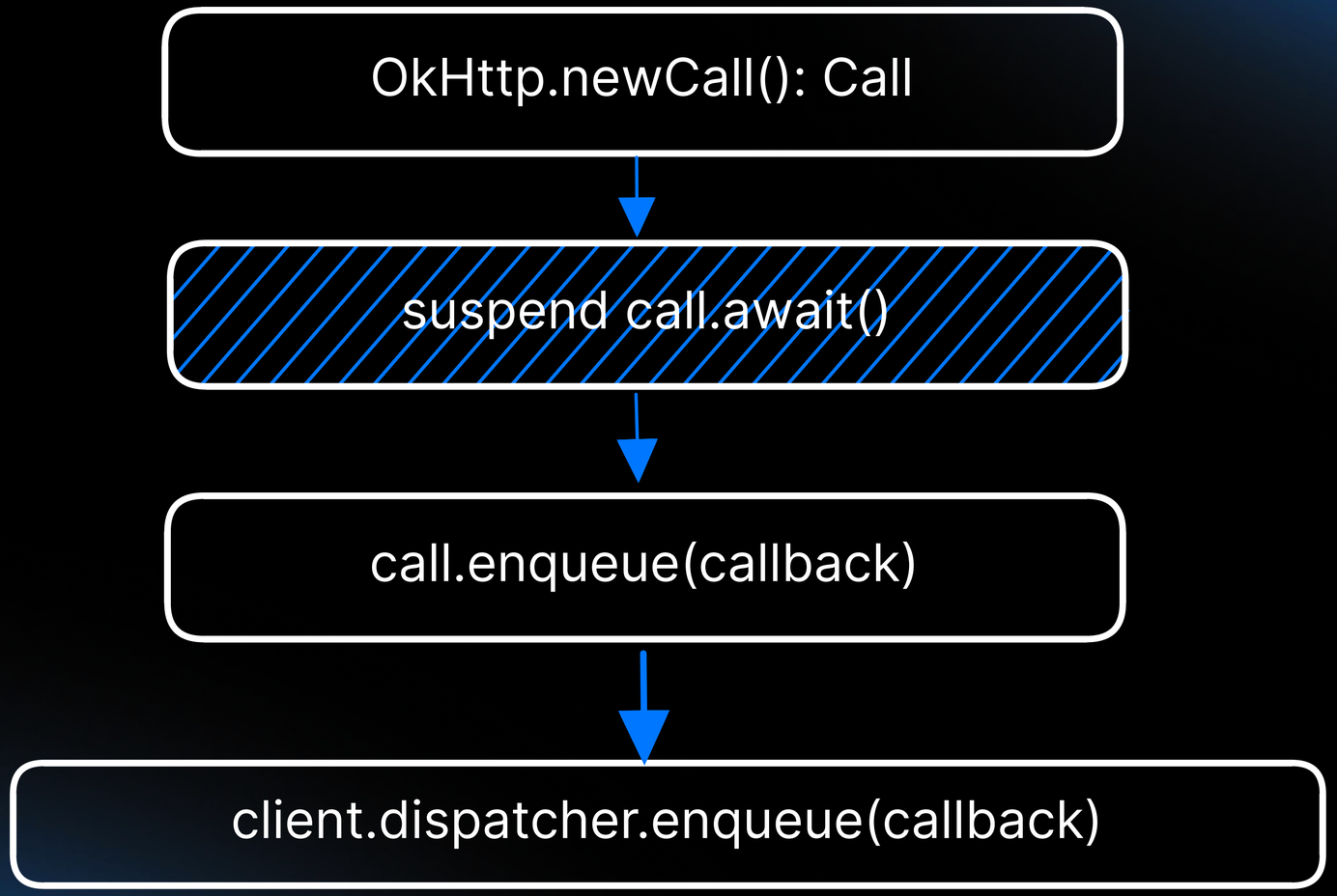
Необходимо затащить аплоад поверх корутин

team

Может, заведется на OkHttp?

```
suspend fun Call.await() = suspendCoroutine { continuation →
    enqueue(object : Callback {
        override fun onResponse(call: Call, response: Response) {
            continuation.resume(response)
        }

        override fun onFailure(call: Call, e: IOException) {
            continuation.resumeWithException(callStack ?: e)
        }
    })
}
```

Но мы все равно встречаемся с Dispatcher

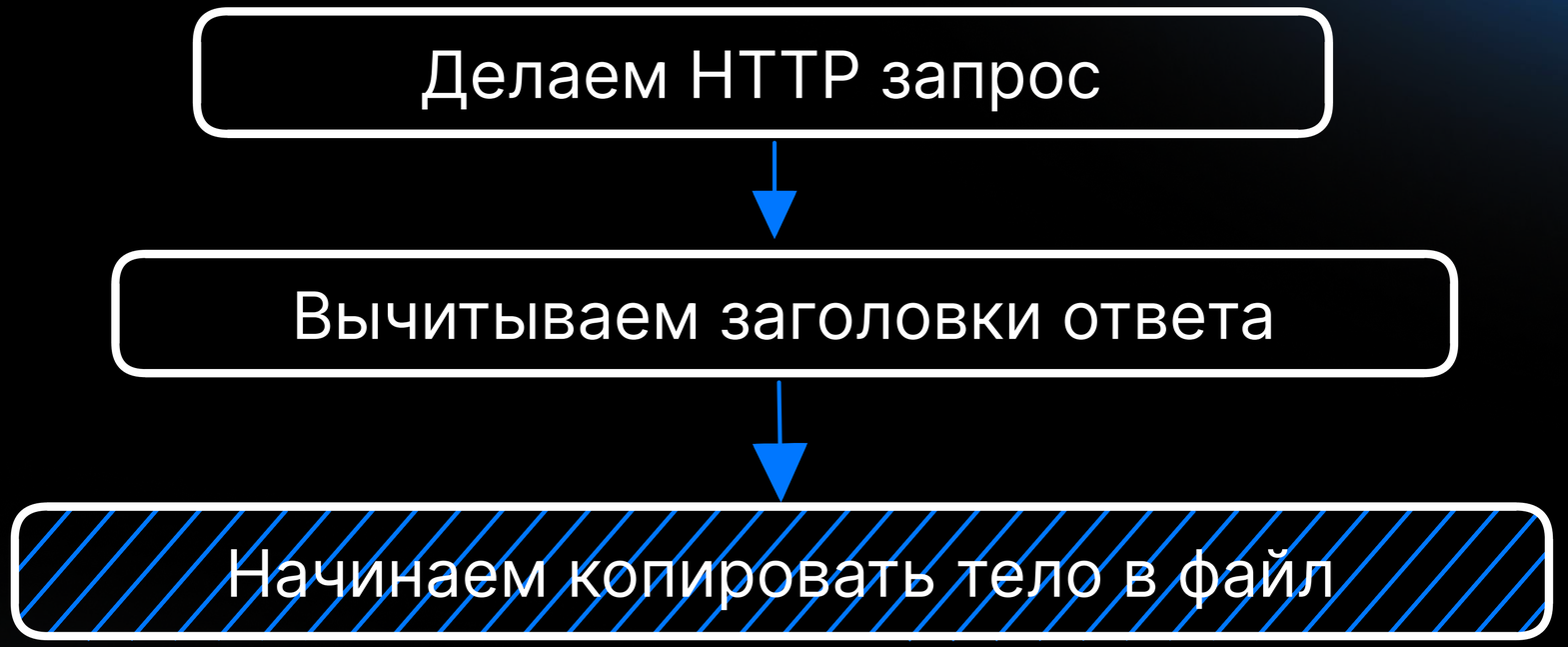


Грузим файлы

А что насчет параллельного скачивания нескольких файлов?

team

Какой у нас флоу



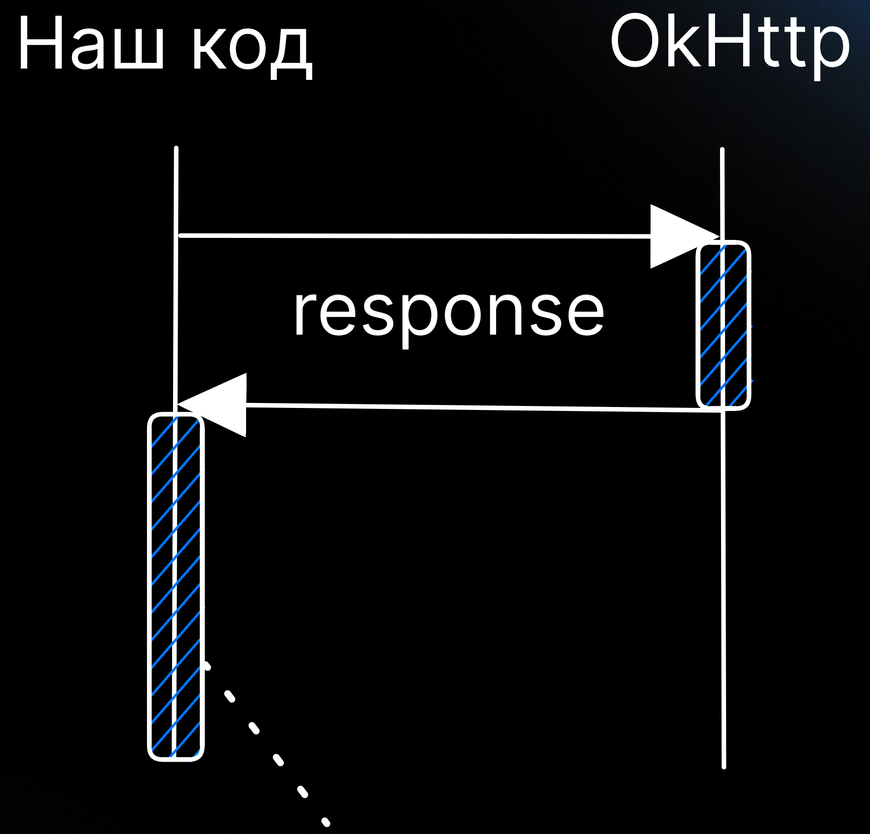
Копирование тела в файл

```
response.body?.byteStream()?.use { input →
    outputFile.outputStream().use { output →
        var bytes = input.read(buffer)
        do {
            output.write(buffer)
            bytes = input.read(buffer)
        } while (bytes ⩾ 0)
        output.flush()
    }
}
```

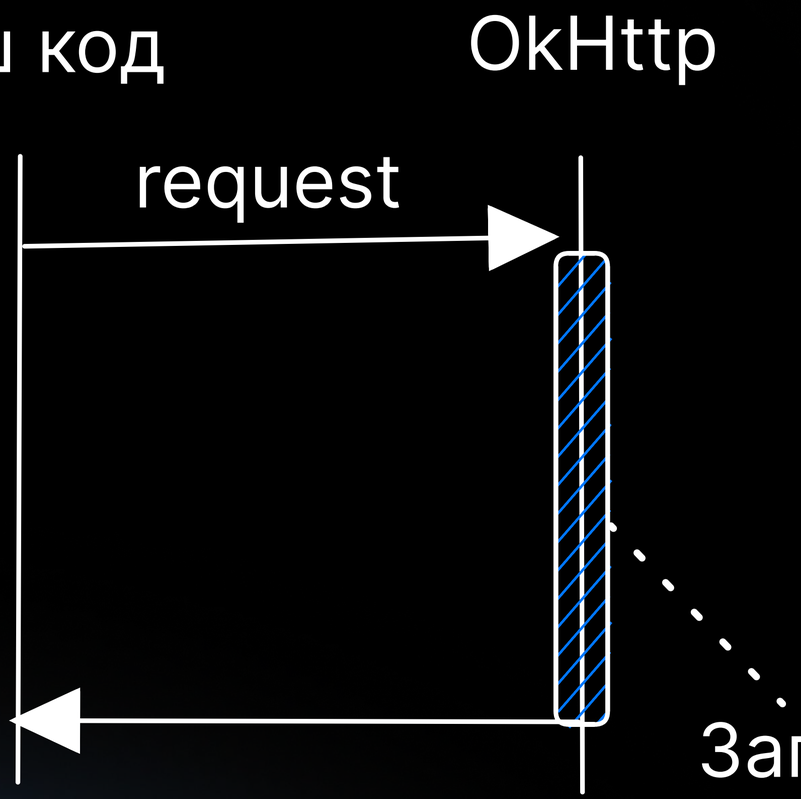
А что, если yeild?

```
response.body?.byteStream()?.use { input →
    outputFile.outputStream().use { output →
        var bytes = input.read(buffer)
        do {
            output.write(buffer)
            yield()
            bytes = input.read(buffer)
        } while (bytes ⩾ 0)
        output.flush()
    }
}
```

yield в массы!



yield в массы!



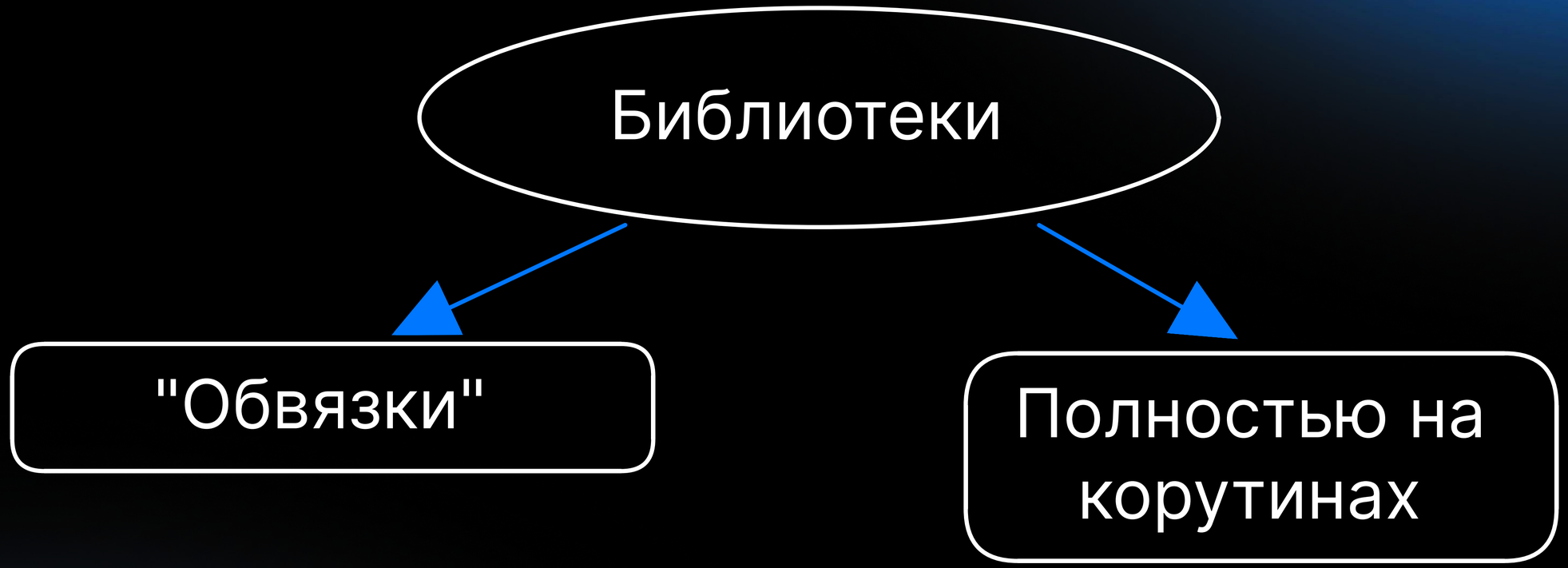
Запись в OutputStream
на стороне OkHttp

Грузим файлы

Берем coroutine-first библиотеку и в прод?

team

На что смотрим?



Почему это не подошло

1. HTTP не распространено
2. Целая библиотека ради одной фичи - оверхед

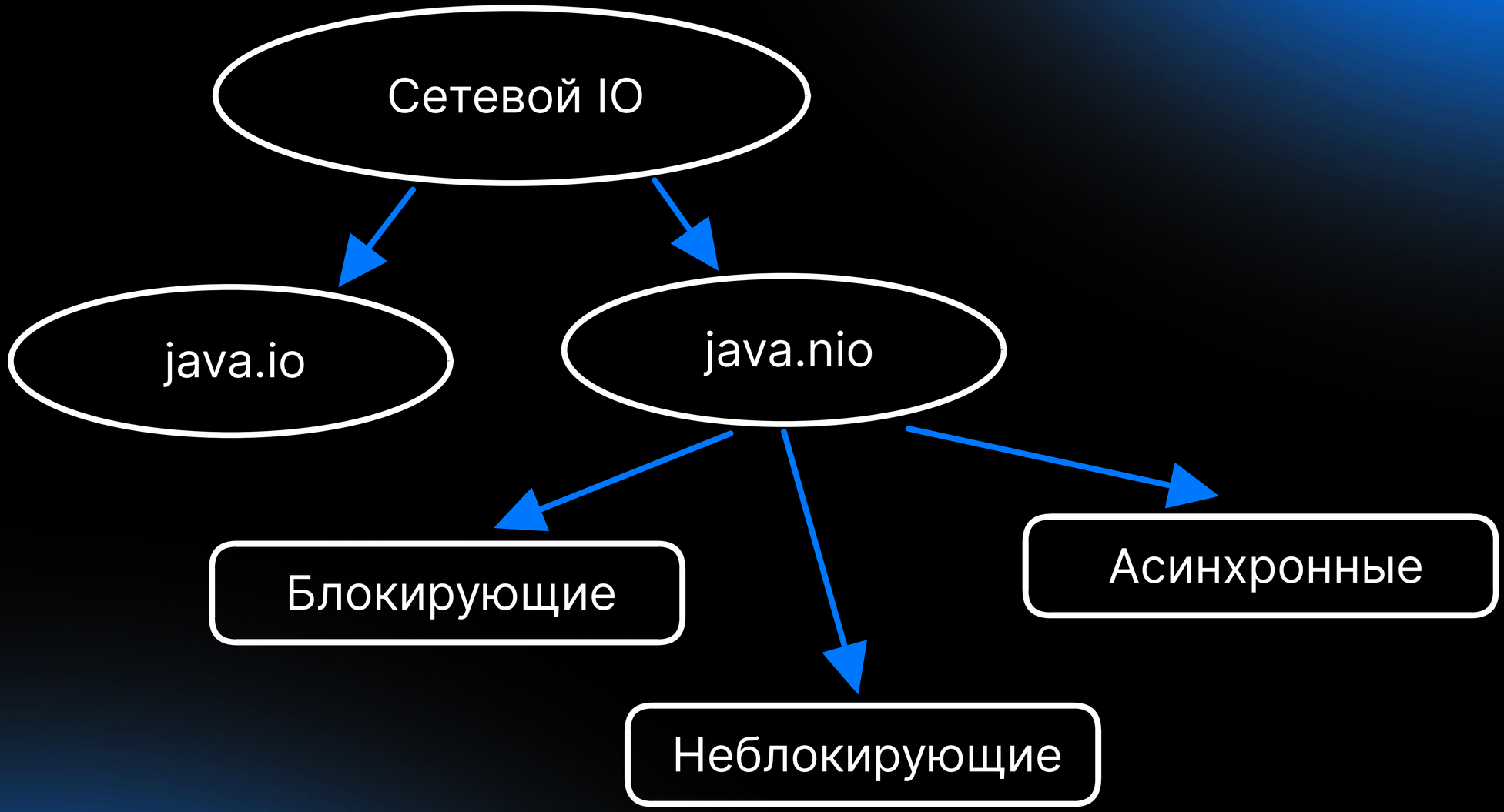
* https://youtrack.jetbrains.com/issue/KTOR-6503

Грузим файлы

Как тогда решать поставленную задачу?

team

Обратимся к Java SDK



Блокирующие каналы

```
FileChannel.open(filepath).use { inputChannel →
    val buffer = ByteBuffer.allocate(SIZE)
    inputChannel.read(buffer)
    println(buffer.convertToString())
}
```

Блокирующие каналы

```
FileChannel.open(filepath).use { inputChannel →
    val buffer = ByteBuffer.allocate(SIZE)
    inputChannel.read(buffer)
    println(buffer.convertToString())
}
```

Асинхронные каналы

```
AsynchronousFileChannel.open(path).use { inputChannel →
    val buffer = ByteBuffer.allocate(SIZE)
    inputChannel.read(buffer, 0, buffer, object : CompletionHandler {
        override fun completed(result: Int, attachment: ByteBuffer) {
            println(buffer.convertToString())
        }

        override fun failed(exc: Throwable, attachment: ByteBuffer) = Unit
    })
}
```

Асинхронные каналы

```
AsynchronousFileChannel.open(path).use { inputChannel →
    val buffer = ByteBuffer.allocate(SIZE)
    inputChannel.read(buffer, 0, buffer, object : CompletionHandler {
        override fun completed(result: Int, attachment: ByteBuffer) {
            println(buffer.convertToString())
        }

        override fun failed(exc: Throwable, attachment: ByteBuffer) = Unit
    })
}
```

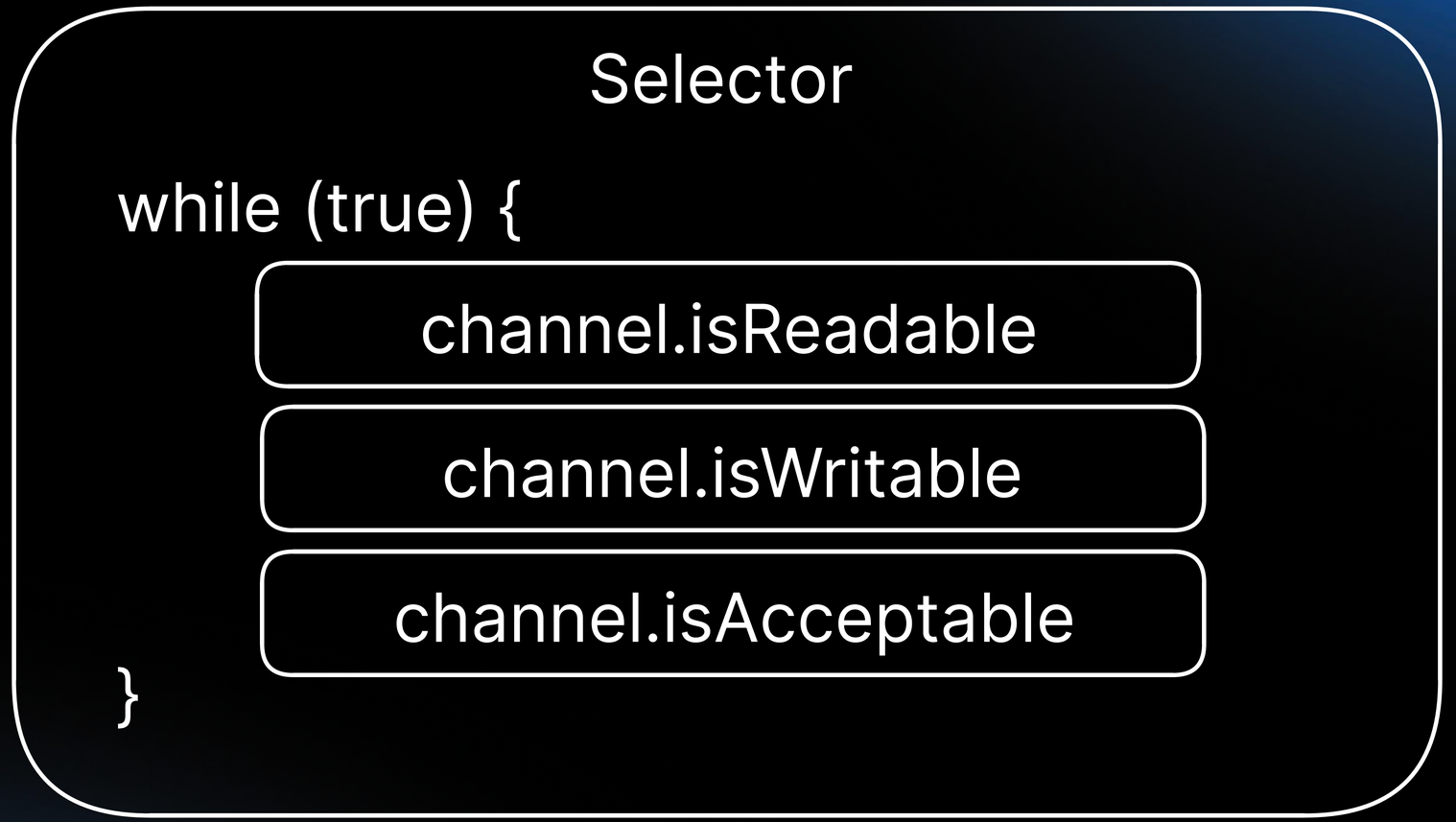
Неблокирующие каналы

```
SocketChannel.open().use { sc →
    sc.configureBlocking(false)
    while (!sc.finishConnect()) { println("waiting connection") }

    sc.write(buffer)
    var receiveReadCount = 0
    while (receiveReadCount == 0) {
        receiveReadCount = sc.read(receivedBuffer)
    }
}
```

Неблокирующие каналы

```
SocketChannel.open().use { sc →
    sc.configureBlocking(false)
    while (!sc.finishConnect()) { println("waiting connection") }

    sc.write(buffer)
    var receiveReadCount = 0
    while (receiveReadCount == 0) {
        receiveReadCount = sc.read(receivedBuffer)
    }
}
```

Selector API



Что нам подойдет?

1. Блокирующие — точно мимо
2. Неблокирующие — раскрываются по-настоящему с Selector API
3. Асинхронные — идеально сочетаются с корутинами

Реализуем колбек для наших extensions

```
object AsyncIOHandlerAny : CompletionHandler {
    override fun completed(result: Any, cont: Continuation<Any>) {
        cont.resume(result)
    }
    override fun failed(ex: Throwable, cont: Continuation<Any>) {
        cont.resumeWithException(ex)
    }
}
```

Пишем сам extension

```
public suspend fun AsynchronousSocketChannel.aWrite(
    buf: ByteBuffer,
    timeout: Long = 0L,
    timeUnit: TimeUnit = TimeUnit.MILLISECONDS,
): Int = suspendCancellableCoroutine<Int> { cont →
    write(buf, timeout, timeUnit, cont, asyncIOHandler())
    closeOnCancel(cont)
}
```

Грузим файлы

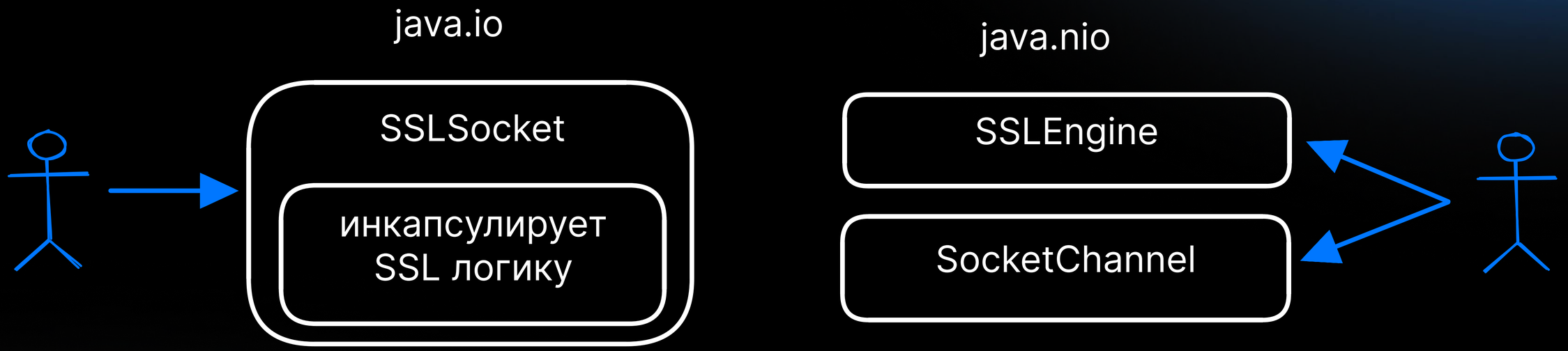
А что по шифрованию трафика?

team

Немного истории

Java Version	Как готовить SSL
1.2	SSLSocket
1.4	Релиз java.nio, SSL не поддержан
1.5	Добавление SSLEngine

java.io vs java.nio



Минусы готовых решений

1

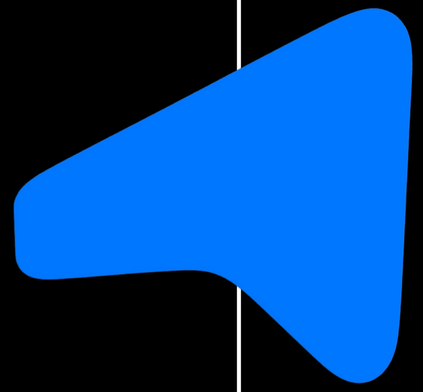
Не подружить с корутинами

2

Нерасширяемое API

3

Попросту не завелось



Значит, вдохновляемся хорошими примерами и делаем решение под себя

* Самый удачная библиотека на просторах гитхаба - https://github.com/marianobarrios/tls-channel

BufferAllocator

```
public interface BufferAllocator {
    public fun allocate(size: Int): ByteBuffer
    public fun free(buffer: ByteBuffer)
}
```

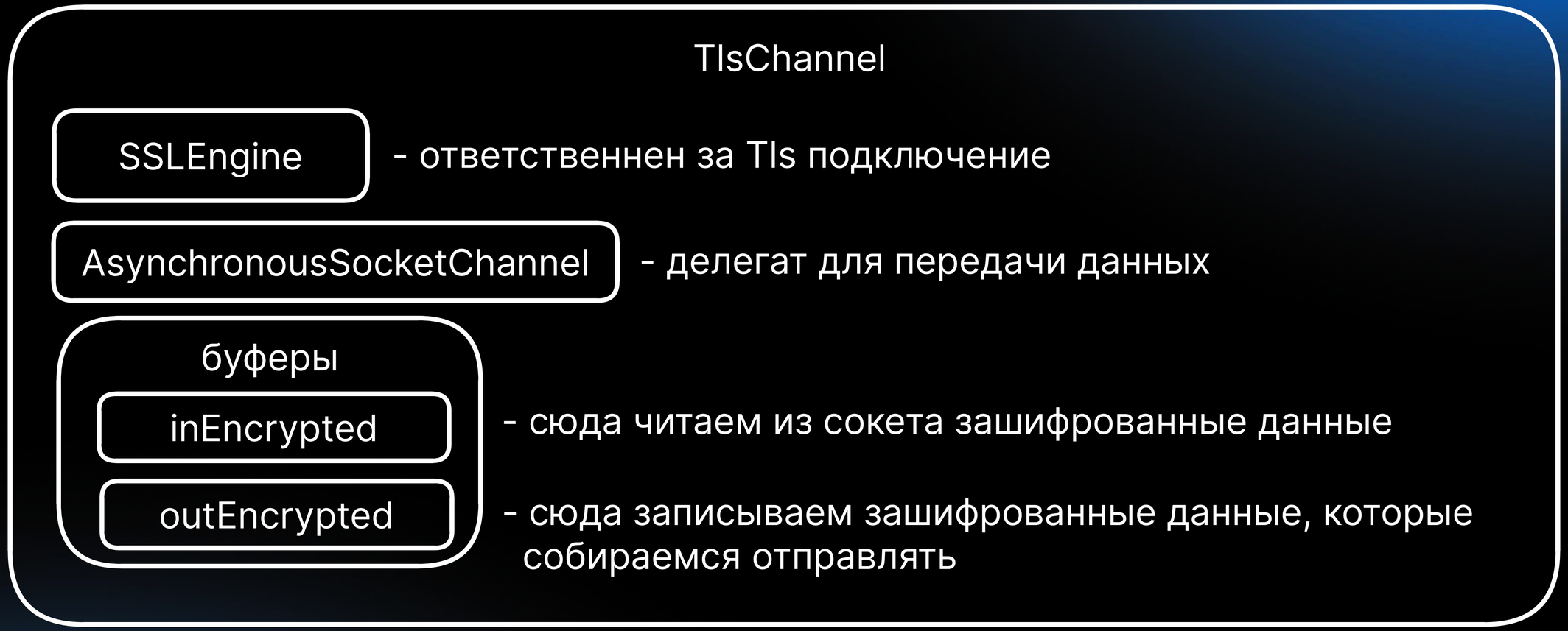
DirectBufferAllocator

```
class DirectBufferAllocator : BufferAllocator {
    val cleanerGetter by declaredMethod("sun.nio.ch.DirectBuffer", "cleaner")
    val cleanMethod by declaredMethod("sun.misc.Cleaner", "clean")

    override fun allocate(size: Int) = ByteBuffer.allocateDirect(size)

    override fun free(buffer: ByteBuffer) {
        val cleaner = cleanerGetter?.invoke(buffer) ?: return
        cleanMethod?.invoke(cleaner)
    }
}
```

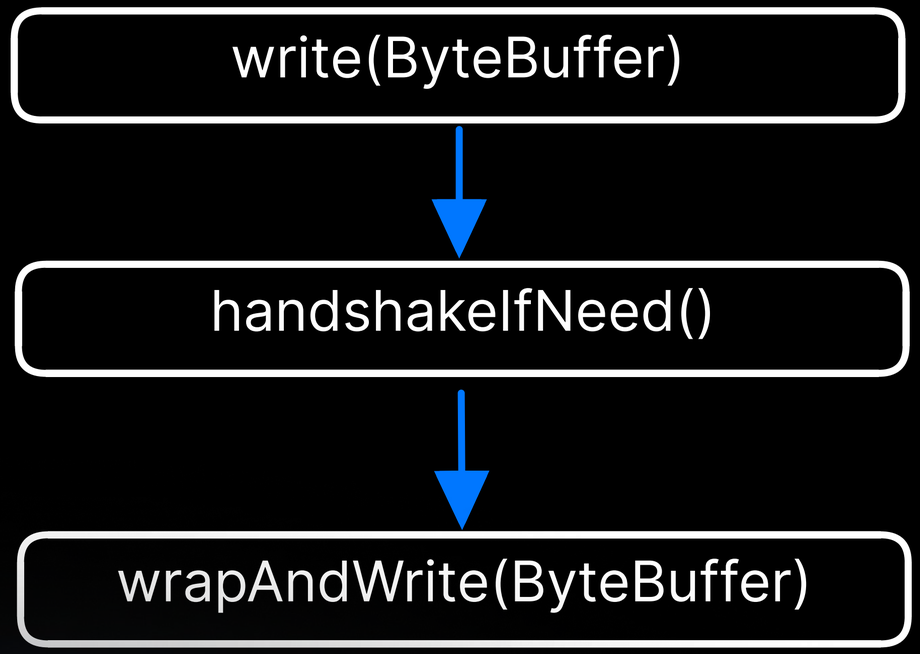
DirectBufferAllocator

```
class DirectBufferAllocator : BufferAllocator {
    val cleanerGetter by declaredMethod("sun.nio.ch.DirectBuffer", "cleaner")
    val cleanMethod by declaredMethod("sun.misc.Cleaner", "clean")

    override fun allocate(size: Int) = ByteBuffer.allocateDirect(size)

    override fun free(buffer: ByteBuffer) {
        val cleaner = cleanerGetter?.invoke(buffer) ?: return
        cleanMethod?.invoke(cleaner)
    }
}
```

TlsChannel


TlsChannel
SSLEngine
- ответственнен за Tls подключение
AsynchronousSocketChannel
- делегат для передачи данных
буферы
inEncrypted
- сюда читаем из сокета зашифрованные данные
outEncrypted
- сюда записываем зашифрованные данные, которые собираемся отправлять

Как записывать данные

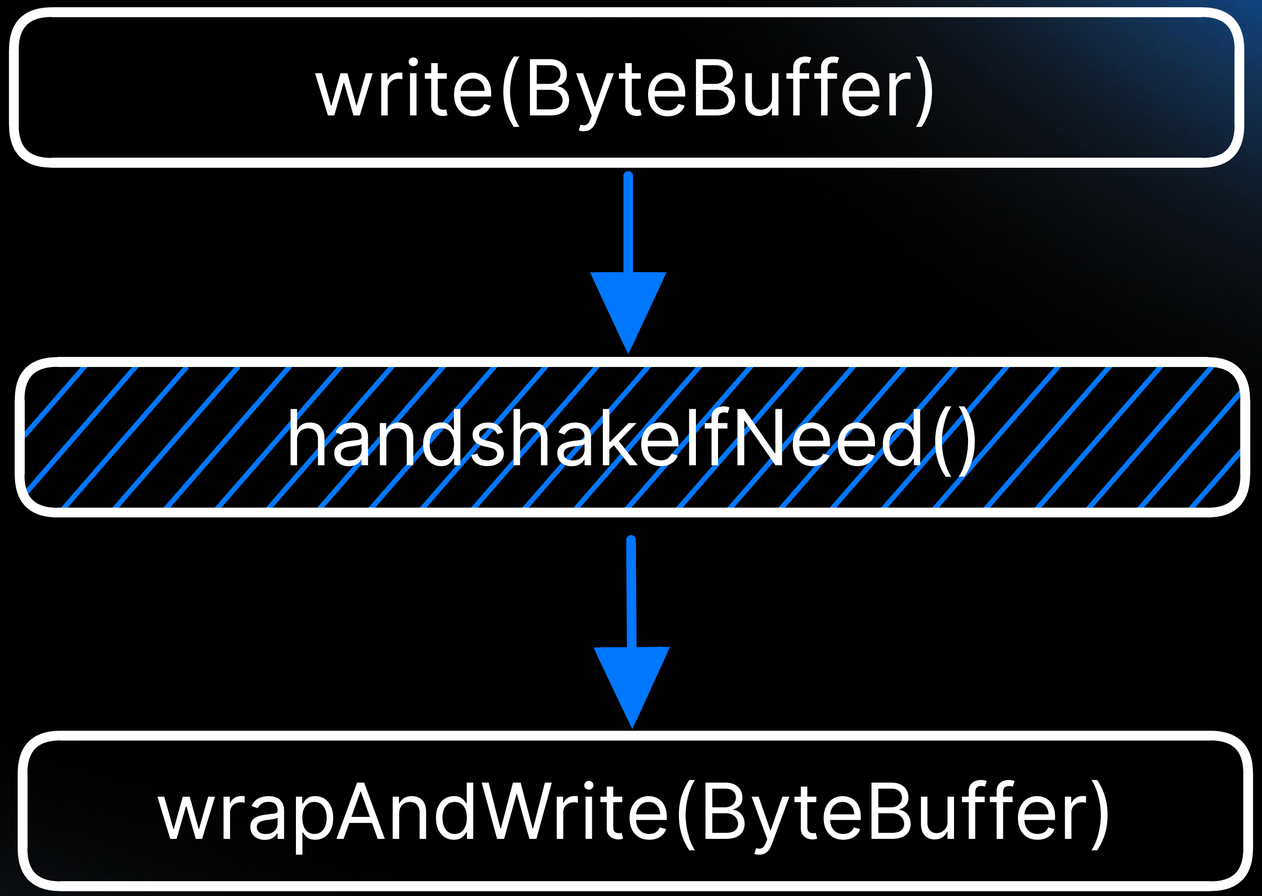


- различные валидации

- в первый раз надо провести SSL handshake

- шифрование и запись в канал делегат

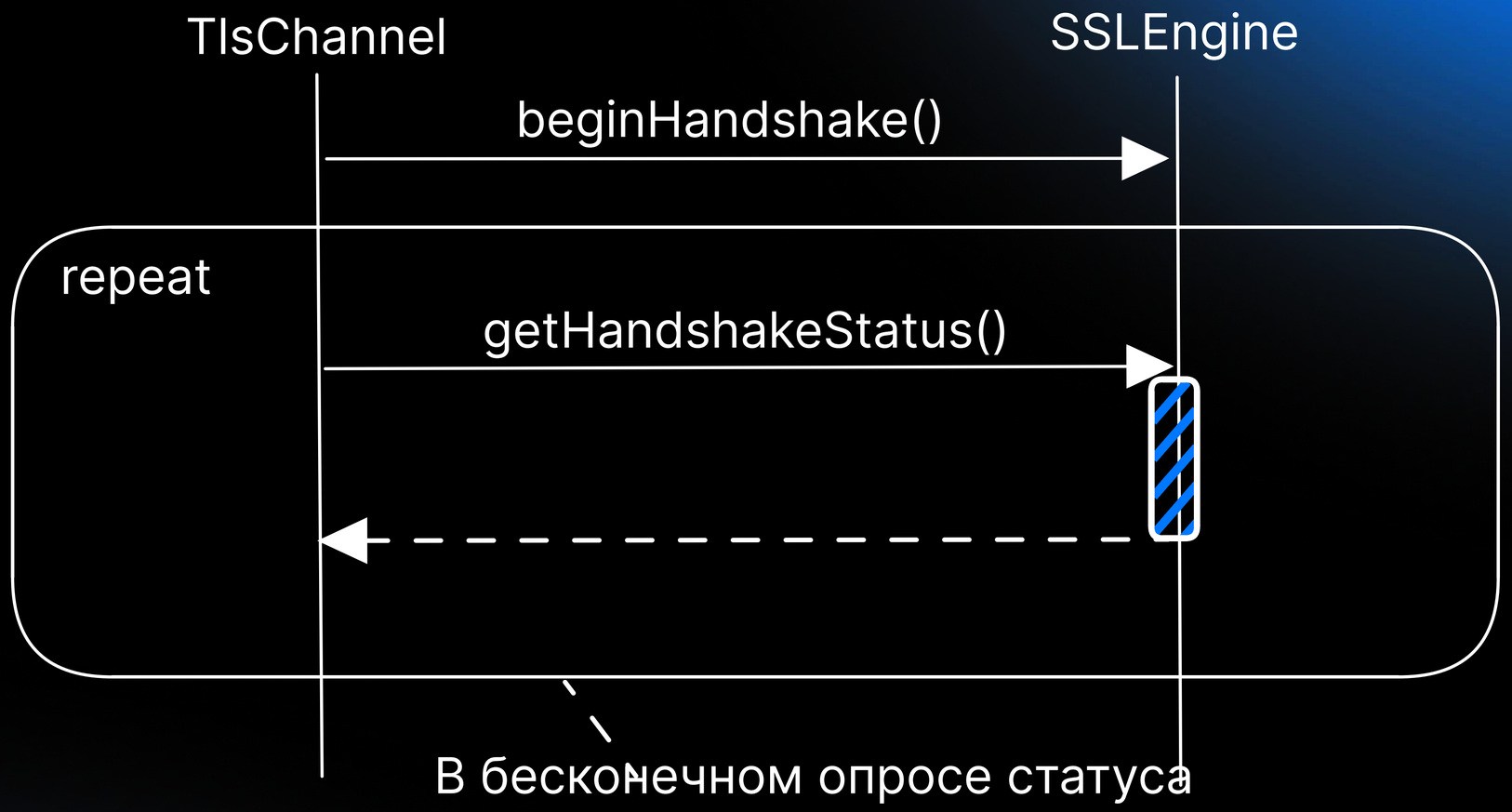
Как записывать данные



Хендшейк

```
val engine = sslContext.createSSLEngine(host, port)
engine.useClientMode = true
engine.beginHandshake()
```

Хендшейк


TlsChannel
SSLEngine
beginHandshake()
repeat
getHandshakeStatus()
В бесконечном опросе статуса
до конца хендшейка

Какие могут быть статусы?

```
HandshakeStatus.NEED_WRAP → {
    wrap(dummyOut)
    delegate.writeToChannel(outEncrypted)
}

HandshakeStatus.NEED_UNWRAP → {
    readAndUnwrap()
    if (bytesToReturn > 0) return
}
```

Какие могут быть статусы?

```
HandshakeStatus.NEED_WRAP → {
    wrap(dummyOut)
    delegate.writeToChannel(outEncrypted)
}

HandshakeStatus.NEED_UNWRAP → {
    readAndUnwrap()
    if (bytesToReturn > 0) return
}
```

Какие могут быть статусы?

```
HandshakeStatus.NOT_HANDSHAKING → {
    return
}

HandshakeStatus.NEED_TASK → runInterruptible {
    sslEngine.delegatedTask?.run()
}

HandshakeStatus.FINISHED → error("Unreachable status")
```

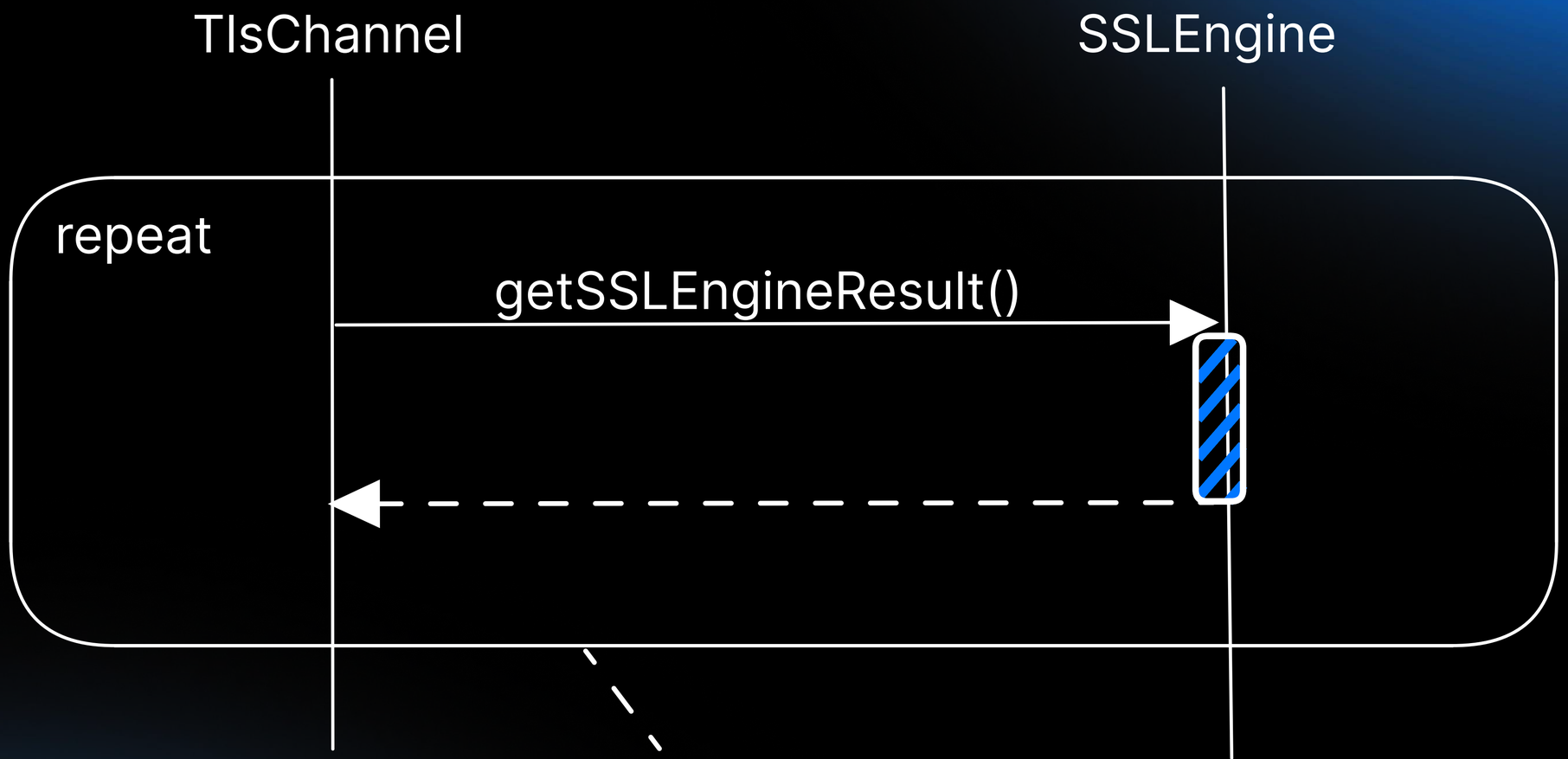
Какие могут быть статусы?

```
HandshakeStatus.NOT_HANDSHAKING -> {
    return
}

HandshakeStatus.NEED_TASK -> runInterruptible {
    sslEngine.delegatedTask?.run()
}

HandshakeStatus.FINISHED -> error("Unreachable status")
```

Какие могут быть статусы?

```
HandshakeStatus.NOT_HANDSHAKING -> {
    return
}

HandshakeStatus.NEED_TASK -> runInterruptible {
    sslEngine.delegatedTask?.run()
}

HandshakeStatus.FINISHED -> error("Unreachable status")
```

Шифруем трафик



В бесконечном опросе статуса
до конца шифрования

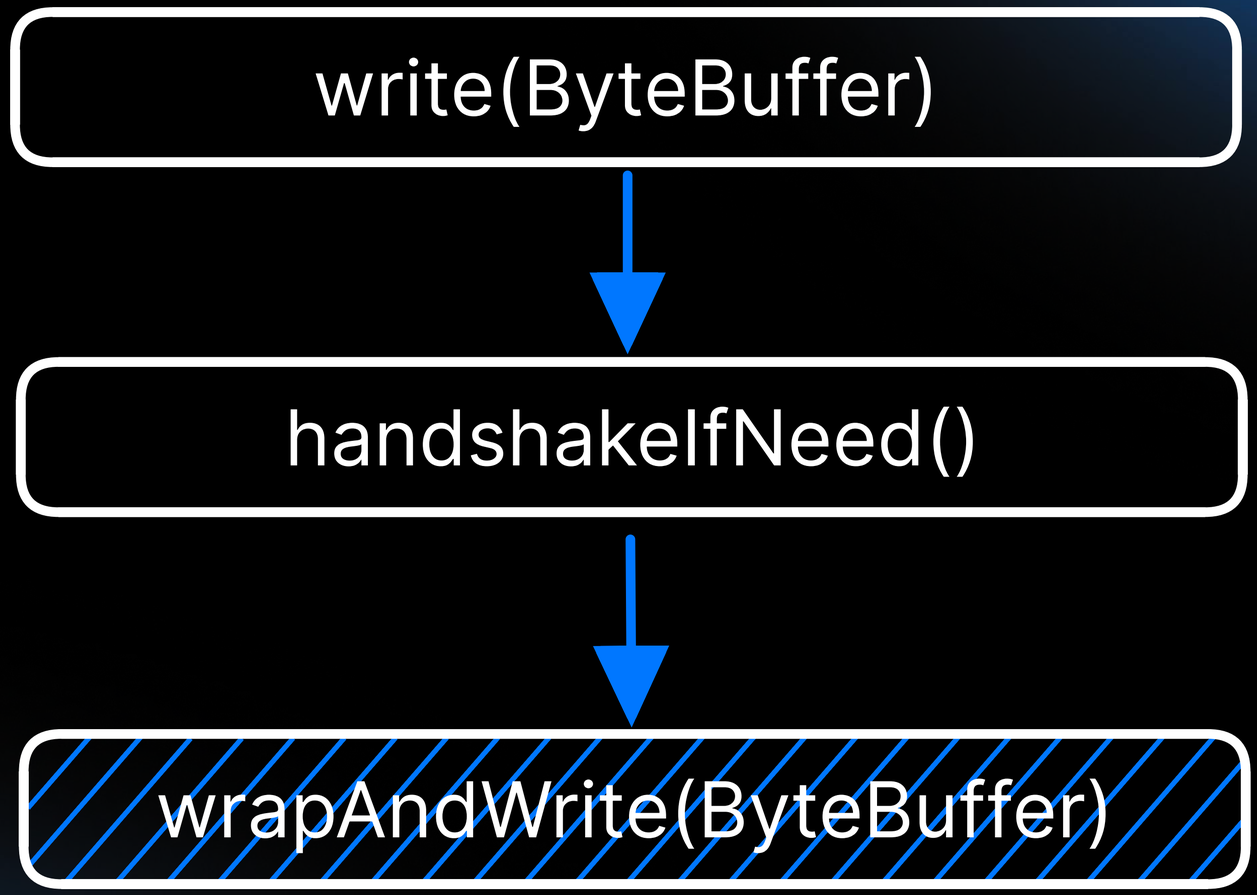
Шифруем трафик

```
suspend fun wrapLoop(source: ByteBuffer): SSLEngineResult {
    while (currentCoroutineContext().isActive) {
        val result = callEngineWrap(source)
        when (result.status) {
            SSLEngineResult.Status.OK,
            SSLEngineResult.Status.CLOSED → return result
            SSLEngineResult.Status.BUFFER_OVERFLOW → outEncrypted.enlarge()
            SSLEngineResult.Status.BUFFER_UNDERFLOW → error("Undefined status")
        }
    }
}
```

Шифруем трафик

```
suspend fun wrapLoop(source: ByteBuffer): SSLEngineResult {
    while (currentCoroutineContext().isActive) {
        val result = callEngineWrap(source)
        when (result.status) {
            SSLEngineResult.Status.OK,
            SSLEngineResult.Status.CLOSED -> return result
            SSLEngineResult.Status.BUFFER_OVERFLOW -> outEncrypted.enlarge()
            SSLEngineResult.Status.BUFFER_UNDERFLOW -> error("Undefined status")
        }
    }
}
```

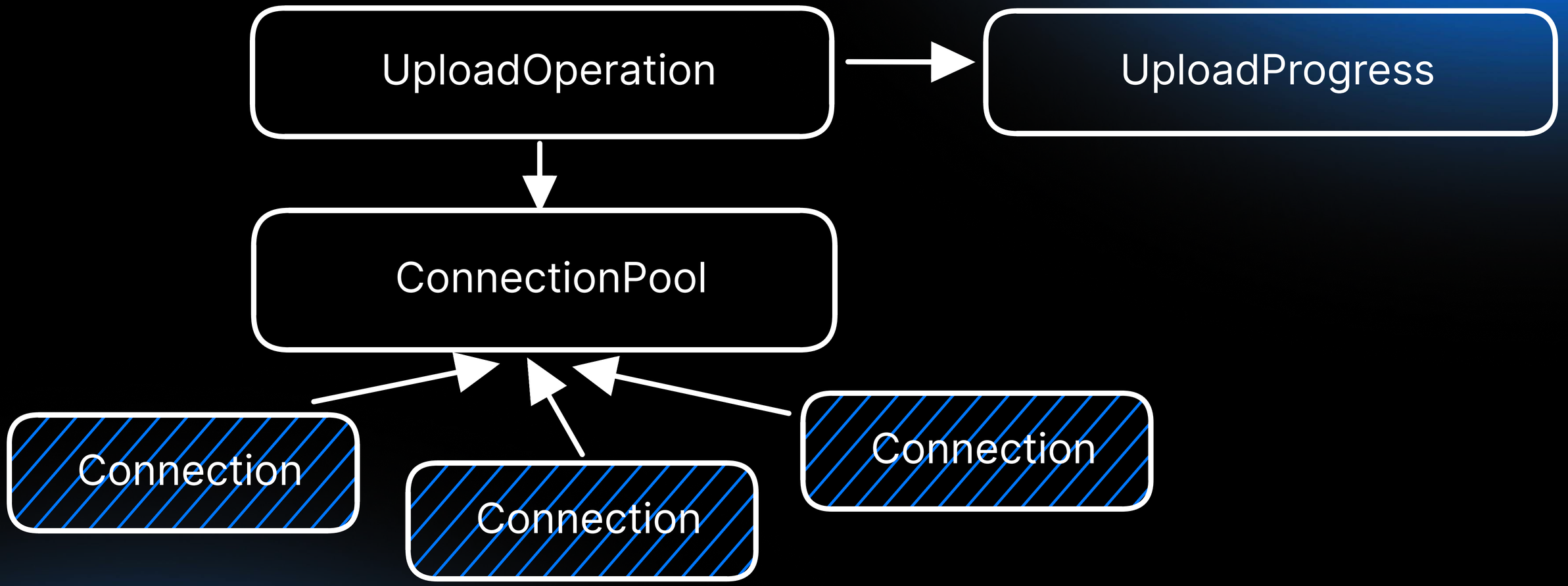
Как записывать данные



Наконец-то пишем в канал-делегат

```
private suspend fun writeWrapped(source: ByteBuffer): Long {
  val bytesToConsume = source.remaining()
  while (currentCoroutineContext().isActive) {
      writeToChannel()
      if (source.remaining() == 0L) return bytesToConsume

      val result = wrapLoop(source)
      if (result.status == SSLEngineResult.Status.CLOSED) {
          return bytesToConsume - source.remaining()
      }
  }
}
```

Время собирать аплоадер



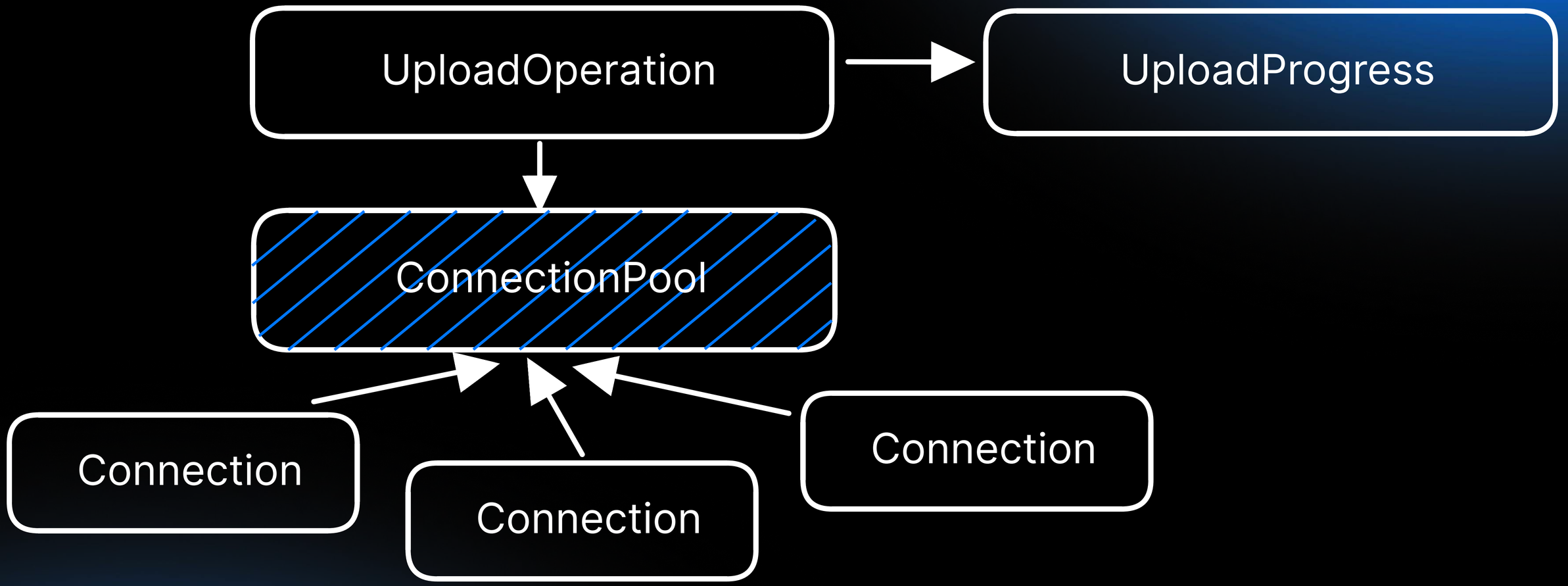
Абстракция Connection

```
public interface Connection {
    public val fileBuffer: ByteBuffer
    public suspend fun connect(host: String)
    public suspend fun write(src: ByteBuffer): Int
    public suspend fun read(dest: ByteBuffer): Int
    public suspend fun close()
}
```

Абстракция Connection

```
public interface Connection {
    public val fileBuffer: ByteBuffer
    public suspend fun connect(host: String)
    public suspend fun write(src: ByteBuffer): Int
    public suspend fun read(dest: ByteBuffer): Int
    public suspend fun close()
}
```

Собираем дальше



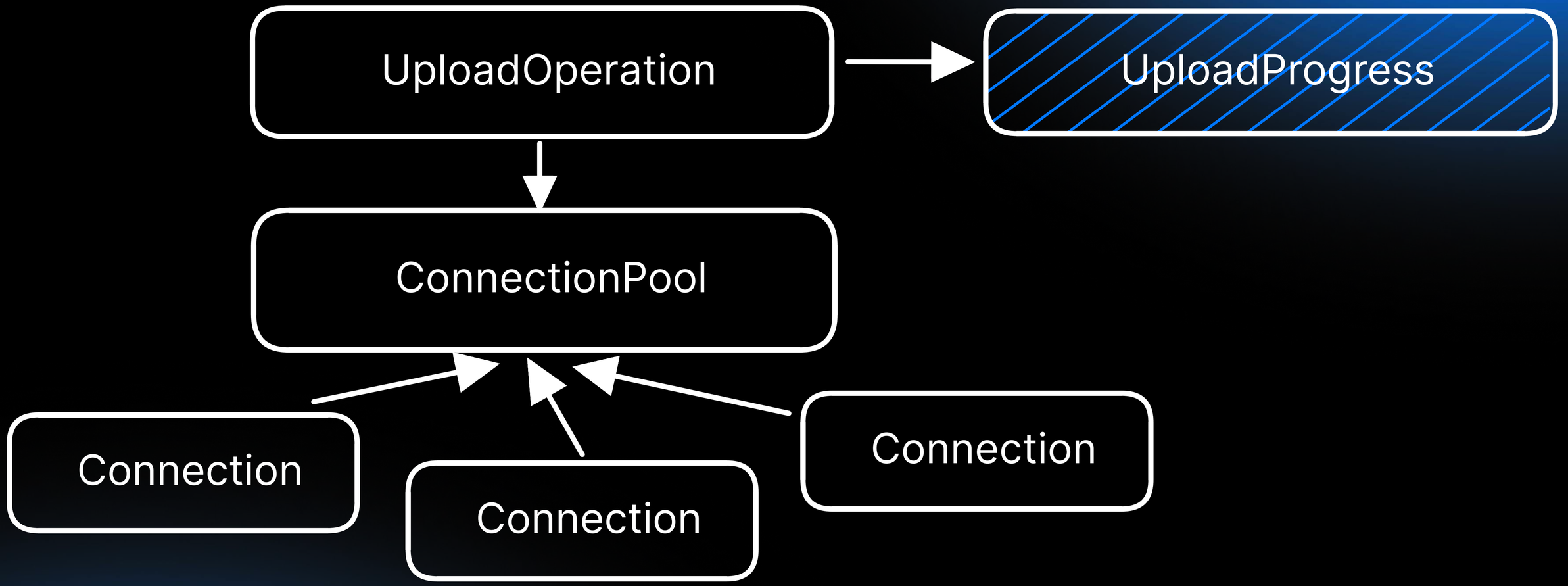
Абстракция ConnectionPool

```
interface ConnectionPool {
    suspend fun acquireConnection(block: AcquireCallback)
    suspend fun releaseConnection(connection: Connection)
    suspend fun close()
}
```

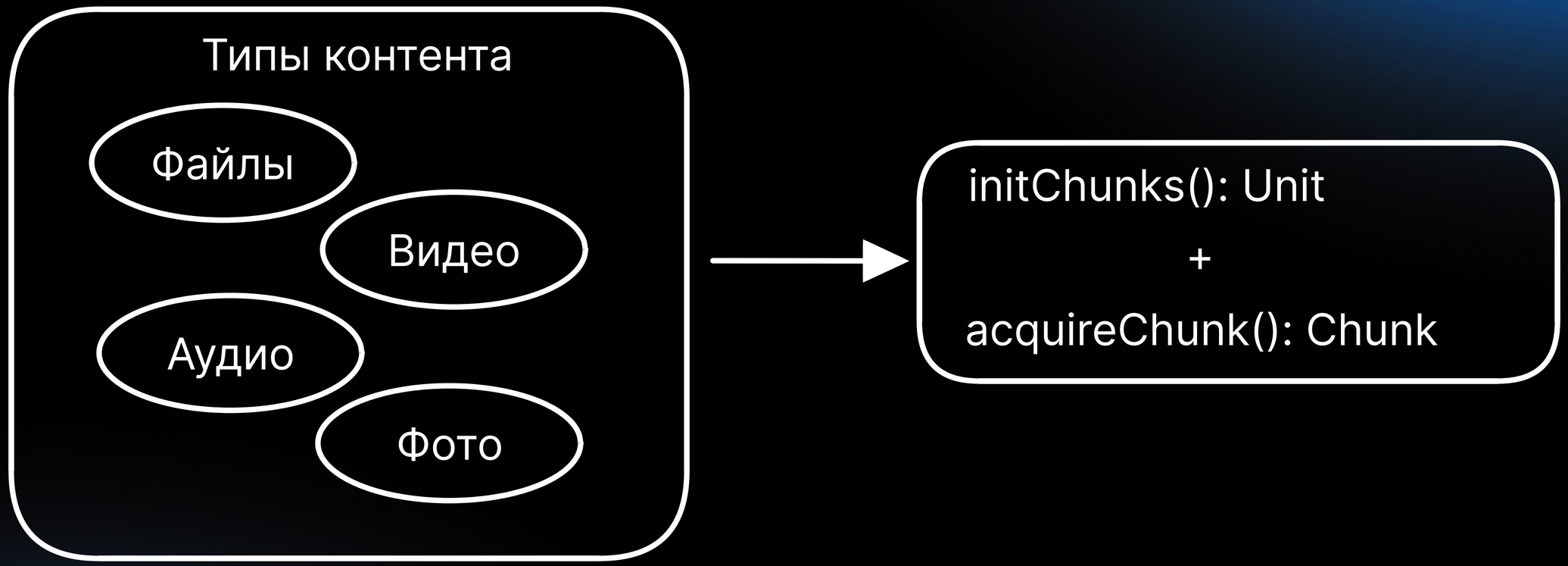
Абстракция ConnectionPool

2G	3G	4G	WIFI
7	7	10	10

Собираем дальше



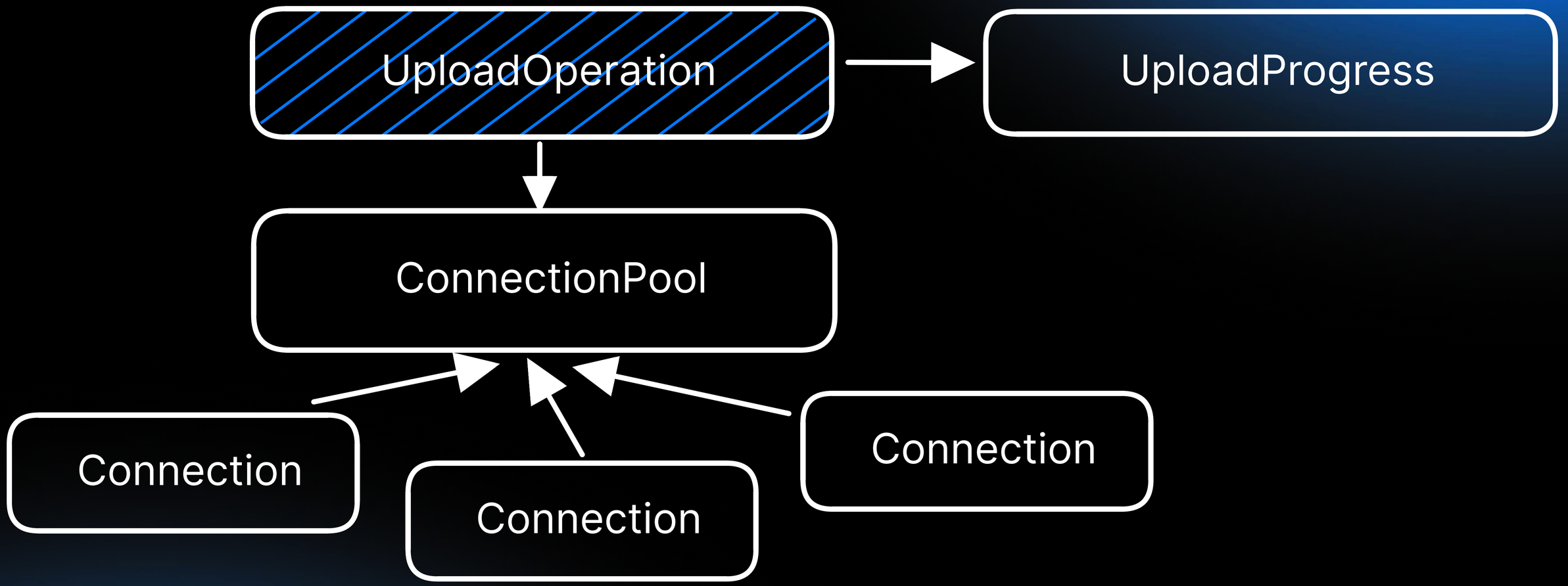
Абстракция UploadProgress



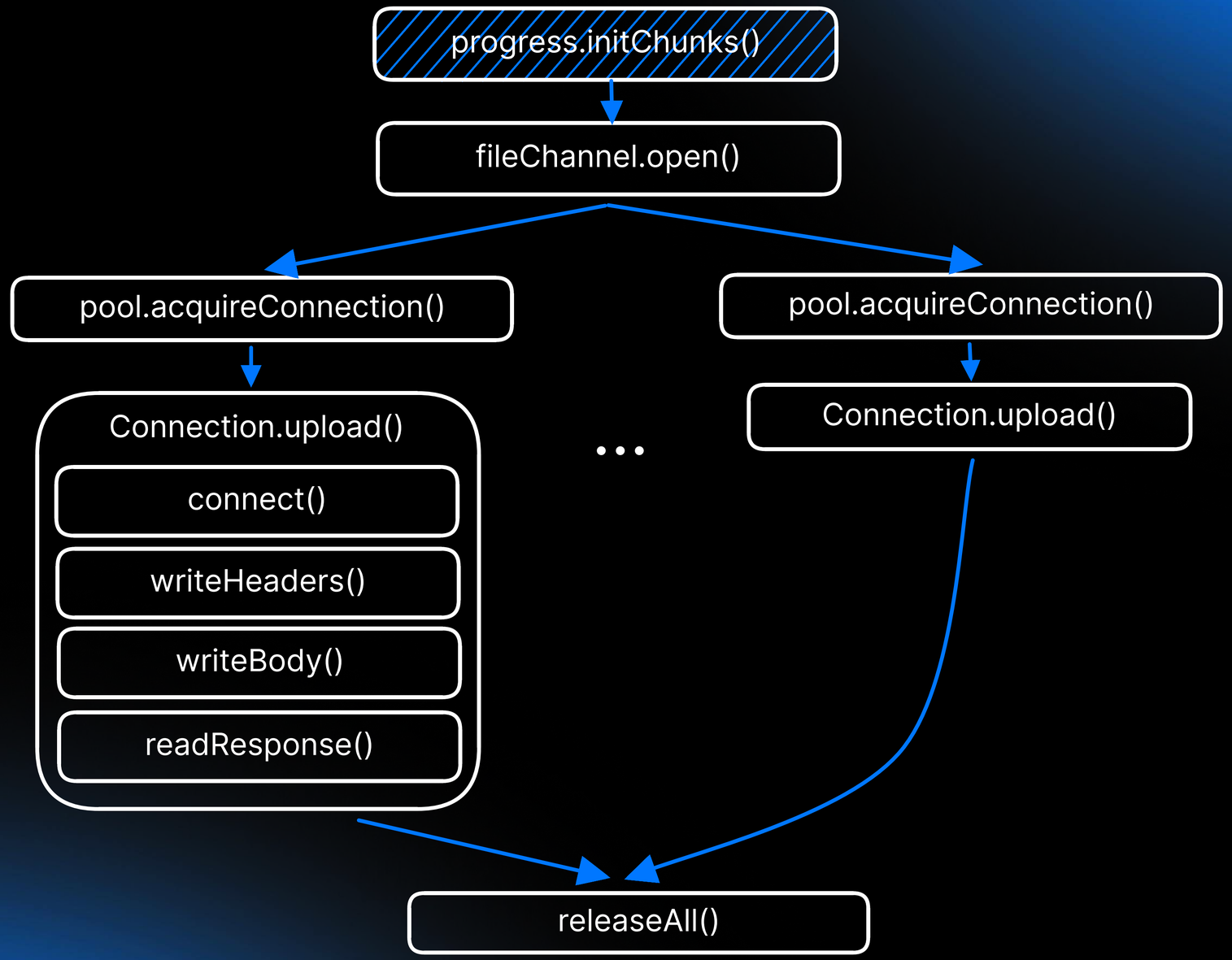
Абстракция UploadProgress

2G	3G	4G	WIFI
16Kb	64Kb	2Mb	2Mb

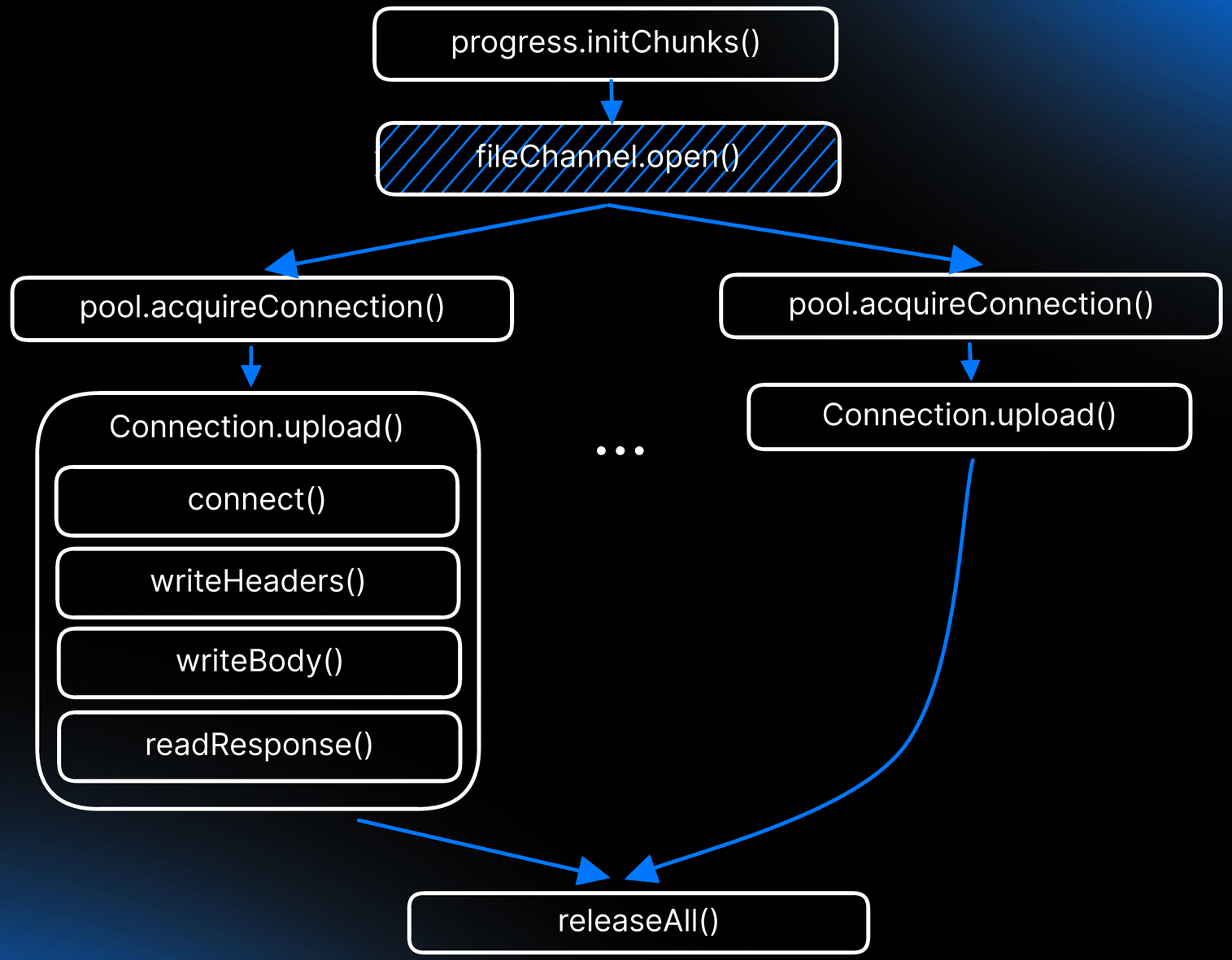
Собираем дальше



Углубляемся в аплоадер



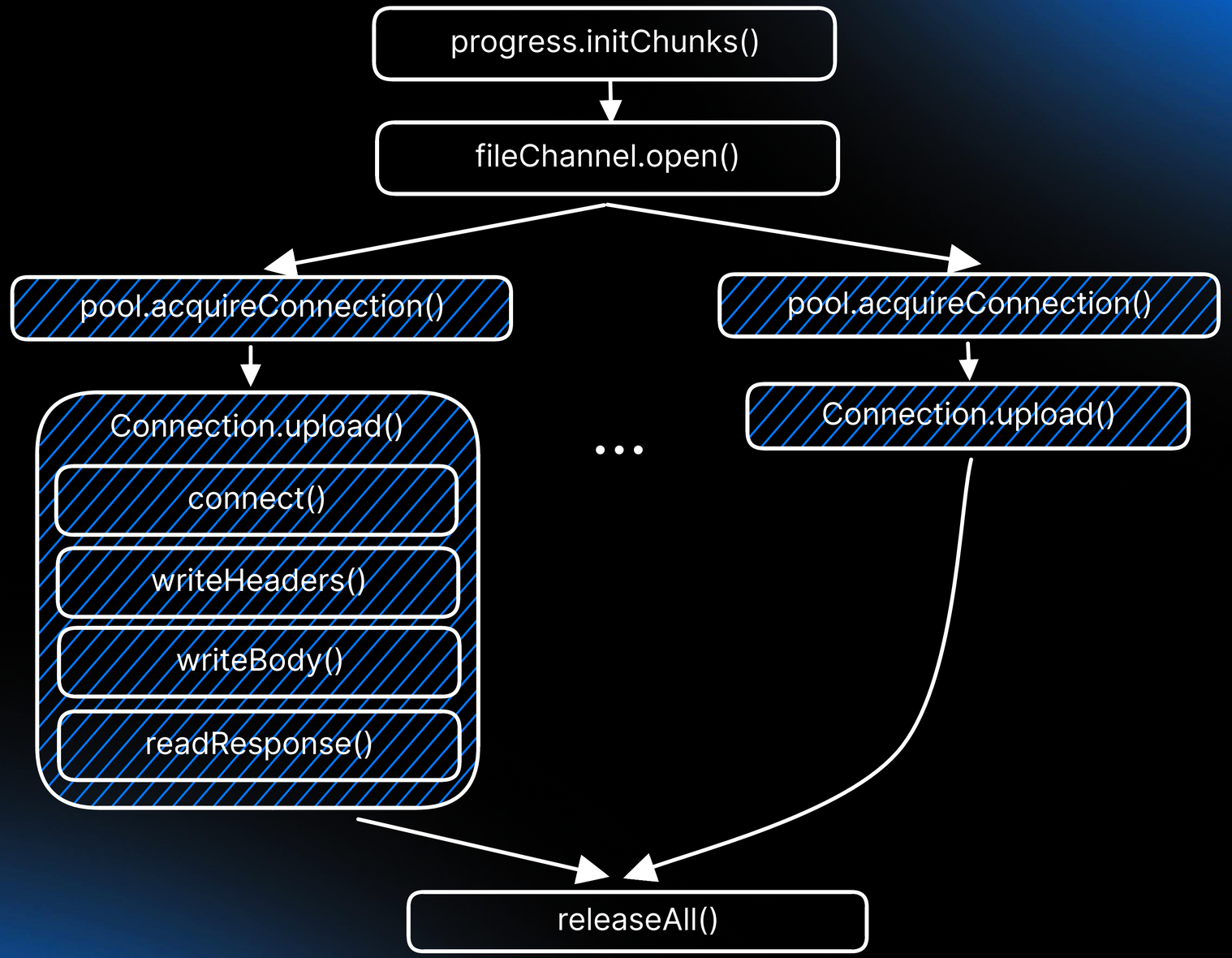
Углубляемся в аплоадер



Открываем канал к файлу

```
runInterruptible {
    AsynchronousFileChannel.open(
        /* file = */ Paths.get(file.path),
        /* options = */ setOf(StandardOpenOption.READ),
        /* executor = */ ioExecutor,
    )
  }
}
```

Углубляемся в аплоадер



Запрашиваем Connection

```
executionScope.launch(uploadJob) {
  connectionPool.acquireConnection { connection →
    val chunk = progress.acquireChunk()
    with(connection) {
      connect(uri.host)
      writeHeaders(chunk)
      writeBody(chunk, fileChannel, onProgress)
      readAndCheckResponse(chunk)
    }
  }
}
uploadJob.children.joinAll()
```

Запрашиваем chunk

```
executionScope.launch(uploadJob) {
  connectionPool.acquireConnection { connection →
    val chunk = progress.acquireChunk()
    with(connection) {
      connect(uri.host)
      writeHeaders(chunk)
      writeBody(chunk, fileChannel, onProgress)
      readAndCheckResponse(chunk)
    }
  }
}
uploadJob.children.joinAll()
```

Начинаем писать чанк на сервер

```
executionScope.launch(uploadJob) {
  connectionPool.acquireConnection { connection →
    val chunk = progress.acquireChunk()
    with(connection) {
      connect(uri.host)
      writeHeaders(chunk)
      writeBody(chunk, fileChannel, onProgress)
      readAndCheckResponse(chunk)
    }
  }
}
uploadJob.children.joinAll()
```

Как именно пишем хедеры

Статичные
- User-Agent
- Content-Type
- Connection
...

Динамические
- Content-Range
- Content-Length

Запекаем в ByteBuffer
и шарим между нашими
Connection's

Для каждого коннекшена
собираем заново
и динамически упаковываем
в ByteBuffer

Стоило того?

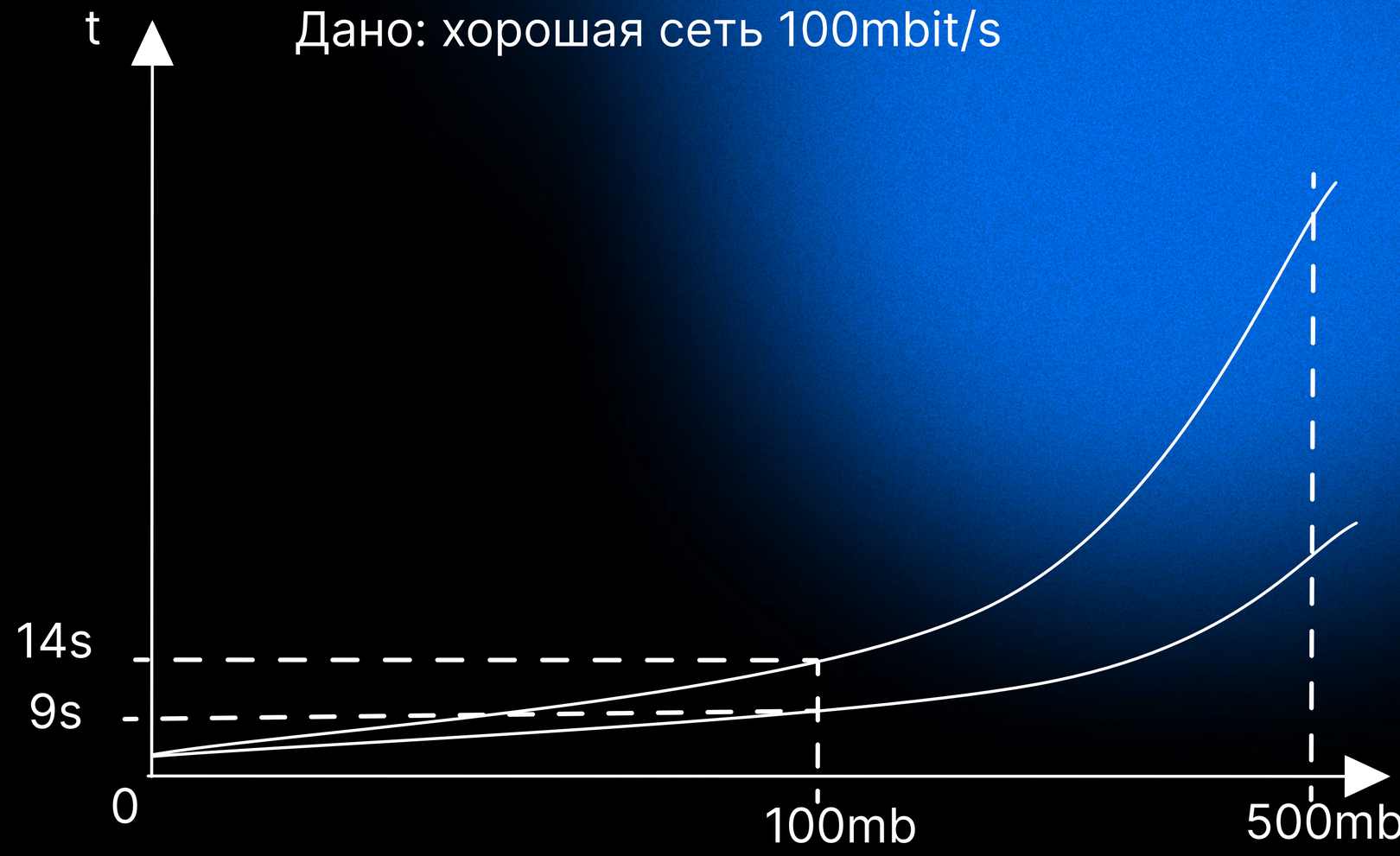
team

Итоги по модулю аплоадера

1. Избавились от легаси на RX от 2017 года, затащили любимые корутины
2. Реализовали все заданные нефункциональные требования
3. Заработали в Single-Core режиме

Итоги по перфу аплоадера

Дано: хорошая сеть 100mbit/s



Грузим файлы

Сможем ли развивать?

team

Проведение АБ

1. Параллелизм Connection’s при аплоаде одного файла

2. Размеры чанков при разных типах сети

Оптимизации

1. Реюз SSLEngine
2. Реализация дуплексного подключения

А что глобально?

1. Аплоадер был отличным плацдармом для сетапа работы с сокетами на корутинах
2. Поняли что вся мощь HTTP библиотек нам не нужна
3. Будем затаскивать параллельное скачивание со склейкой на клиенте

Грузим файлы

Готов ответить на вопросы

team